CSCI 340, Fall 2020
Instructor: Simina Fluture, PhD

340 Project 2 – FALL 2020

The project must be done individually. No exceptions.

You are asked to synchronize the threads of the following story <u>using semaphores</u> and operations on semaphores. <u>Do NOT use busy waiting.</u>

You have the following choices:
1. You can submit a pseudo-code implementation:

Due Date: Wed. Dec 9 (no late submission penalty until Thu. Dec 10. After Dec. 10 no submission will be considered for grading)

Or

2. You can submit a java-code implementation; in that case the project weight will be 7% instead of 6% (you get 1% EC on project 2's weight). I wish I could give you more but too many students are plagiarizing.

Due Date: Wed. Dec 9 (no late submission penalty until Sun. Dec 13. After Dec. 13 no submission will be considered for grading)

<u>DO NOT SUBMIT BOTH TYPES OF IMPLEMENTATION: EITHER YOU SUBMIT THE PSEUDO-CODE, OR YOU SUBMIT THE JAVA-CODE.</u>

Directions: Synchronize the students and teacher in the context of the problem described below. Please refer to and read the project notes carefully, tips and guidelines before starting the project.

****

The project must be done individually - no exceptions. Plagiarism is not accepted. Receiving and/or giving answers and code from/to other students enrolled in this or a previous semester, or a third source including the Internet is academic dishonesty subject to standard University policies.

A SCHOOL DAY DURING COVID

During this COVID time students at PS1111 have to follow strict rules. After a student wakes up and gets ready for a new school day, (s)he will complete a Health Questionnaire (simulate this using a sleep of random time). Next s(he) will commute to school. (sleep of random time).

Once arrived at school, student(s) will **wait** in the schoolyard to be called by the teacher (the teacher will call each of them). Once called by the teacher, before

entering the classroom, students must wash their hands. They will head to the restrooms. There are **two restrooms**, one for "girls" and one for "boys." The capacity of the restroom is three. You can decide if a student is a boy or a girl using a random number or you can consider that students with an odd id are boys while the others are girls. Students will **wait** their turn to use the restrooms.

By the time a student gets to class, if the class is already in session, the student(s) will leave for a while (**sleep of random time**) and walk around the campus and come back later on. If the class is not in session yet, student(s) will **wait** for the teacher to arrive and enter the auditorium.

Once the class is in session, students will immediately get bored and cannot **wait** for the class to end. The teacher will teach (simulate it using **sleep for a fixed interval of time**) and when done, he will let the students know that the class ended. Student(s) will leave the classroom and hurry to have some fun between classes (sleep of random time). (Note: us an implementation similar to the one for source graph. The teacher will signal one student only)

Once having arrived at the school the teacher will let students in. During the day, he will teach 2 periods. Each class takes a fixed amount of the time period. Between any two classes there is a break.

The school closes after the two classes end. At the end of school day, students leave the school. Each student will **wait** another student; they will leave in decreasing order of their name or their ID.

The teacher will **wait** until the last student leaves and after that he will terminate as well.

A daily report with information about what classes and when each student attended throughout the day must be displayed. It can be displayed by the teacher before terminating or in the main method.

Something like:

Student Name Total # of attended classes Class Name (#) Period number.

Your program should implement two types of threads:

Student (there are many students)
Teacher (there is only 1 teacher)

The number of students should be read as command line arguments: *e.g.* –s <int> with default values of 13.

CSCI 340, Fall 2020
Instructor: Simina Fluture, PhD

1. Pseudo-code implementation

You are asked to synchronize the threads of the story using semaphores and operations on semaphores. Do NOT use busy waiting.

Use pseudo-code similar to the one used in class (NOT java pseudo-code).

Mention the operations that can be simulated by a fixed or random sleep_time.

Your documentation should be clear and extensive.

Explain the reason for each semaphore that you used in your implementation. Explain each semaphore type and initialization.

Discuss the possible flows of your implementation. Deadlock is not allowed.

2. Java-code implementation

Using Java programming, synchronize the threads, in the context of the problem. Closely follow the implementation requirements. The synchronization should be implemented through Java semaphores and operations on semaphores (acquire and release)

For Mutual Exclusion implementation use **Mutex semaphores**, not volatile variables.

For semaphore constructors, use ONLY:

Semaphore(int permits, boolean fair)
Creates a Semaphore with the given number of permits and the given fairness setting.

In methods use ONLY: acquire(), release();

You can also use:

getQueueLength()

Returns an estimate of the number of threads waiting to acquire.

hasQueuedThreads()

Queries whether any threads are waiting to acquire.

DO NOT USE ANY OF THE OTHER METHODS of the semaphore's class, besides the ones mentioned above.

CSCI 340, Fall 2020
Instructor: Simina Fluture, PhD

Any wait must be implemented using P(semaphores) (acquire).

Any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.

Document your project and explain the purpose and the initialization of each semaphore.

DO NOT use synchronized methods (beside the operations on semaphores).

Do NOT use wait( ), notify( ) or notifyAll( ) as monitor methods. Whenever a synchronization issue can be resolved use semaphores and not a different type of implementation.

You should keep the concurrency of the threads as high as possible, however the access to shared structures has to be done in a Mutual Exclusive fashion, using a mutex semaphore.

Many of the activities can be simulated using the sleep(of a random time) method.

Use appropriate System.out.println( ) statements to reflect the time of each particular action done by a specific thread. This is necessary for us to observe how the synchronization is working.

Submission similar to project1. Name your project:
YourLastname_Firstname_CS340_p2

Upload it on Blackboard.

- Do not submit any code that does not compile and run. If there are parts of the code that contain bugs, comment it out and leave the code in. A program that does not compile nor run will not be graded.
- Closely follow all the requirements of the project's description.
- The main method is contained in the main thread. All other thread classes must be manually created by either implementing the Runnable interface or extending the Thread class. Separate the classes into separate files (do not leave all the classes in one file, create a class for each type of thread). DO NOT create packages.

- The project asks that you create different types of threads. There is more than one instance of a thread. No manual specification of each thread's activity is allowed (e.g. no Passenger5.goThroughTheDoor())
- Add the following lines to all the threads you make:

```
        public static long time = System.currentTimeMillis();

   public void msg(String m) {
      System.out.println("["+(System.currentTimeMillis()-time)+"] "+getName()+":
"+m);
   }
```

- It is recommended that you initialize the time at the beginning of the main method, so that it is unique to all threads.

  There should be output messages that describe how the threads are executing. Whenever you want to print something from a thread use: msg("some message about what action is simulated");

- NAME YOUR THREADS. Here's how the constructors could look like (you may use any variant of this as long as each thread is unique and distinguishable):

```
// Default constructor
public RandomThread(int id) {
    setName("RandomThread-" + id);
}
```

- Design an OOP program. All thread-related tasks must be specified in its respective classes, no class body should be empty.

- DO NOT USE System.exit(0); the threads are supposed to terminate naturally by running to the end of their run methods.

- Javadoc is not required. Proper basic commenting explaining the flow of the program, self-explanatory variable names, correct whitespace and indentations are required.

Setting up project/Submission:

For those that use Eclipse:

Name your project as follows: LASTNAME_FIRSTNAME_CSXXX_PY, where LASTNAME is your last name, FIRSTNAME is your first name, XXX is your course, and Y is the current project number.

For example: Doe_John_CS340_p2

PLEASE UPLOAD YOUR FILE ON BLACKBOARD IN THE CORRESPONDING COLUMN.